Skład grupy:
Krzysztof Razik 235699
Jakub Majewski 238902

Prowadzący zajęcia:
dr inż. Dariusz Banasiak

Projekt UCiSW 2

Uproszczona symulacja fizyki ziaren piasku reprezentowanych przez piksele na ekranie

KONSPEKT DOKUMENTU

I. Etapy projektu - Założenia projektowe

Celem projektu było stworzenie prostego programu symulującego fizykę związaną z piaskiem, napisaną pod płytkę Spartan 3 (S3E Starter Kit) w języku VHDL. Aplikacja pozwalała by na stawianie oraz usuwanie piasku na danym obszarze wyświetlanym na monitorze VGA. Program miałby być sterowany przez klawiaturę PS2.

1. Sterownik vga

W pierwszym etapie stworzymy moduł, który pozwoli na obsługę monitora vga. Dzięki niemu będziemy mogli manipulować pikselami znajdującymi się na ekranie.

2. Obsługa klawiatury

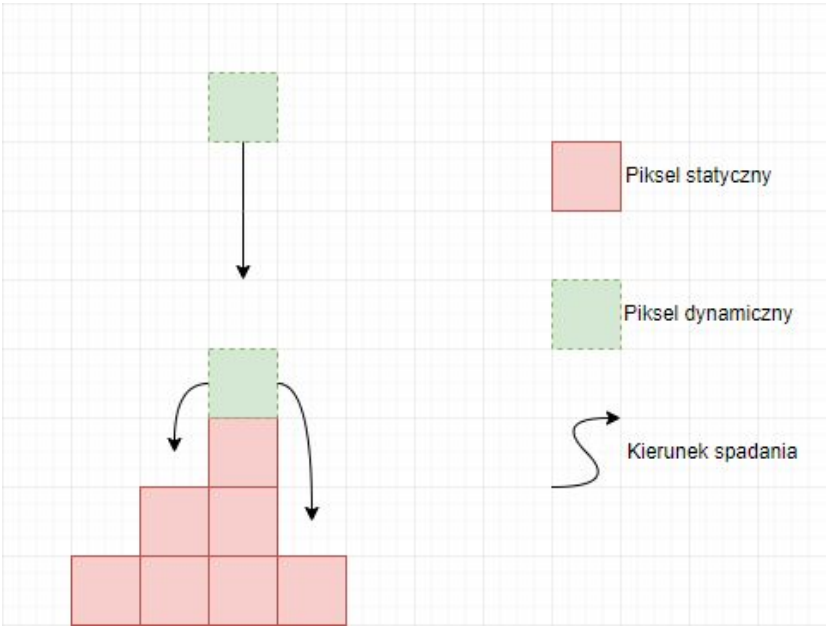
Następnie stworzymy moduł obsługujący kursor na ekranie, którym będzie można sterować przy użyciu klawiatury.

3. Rysowanie pikseli

Informacje i położeniu pikseli będziemy przechowywać w tablicy o dużym rozmiarze. Każdy z pikseli będzie na początku pojawiał się na pozycji równej pozycji kursora. Wyrenderowane piksele będą nieruchome (statyczne). Po tym etapie projekt będzie przypominać prosty program graficzny z możliwością rysowania pikseli na ekranie.

4. Moduł symulujący fizykę piasku

Wprawimy piksele z poprzedniego kroku w ruch. Początkowo jedyną siłą jaka będzie działać na piasek to grawitacja. W ramach oszczędzania mocy obliczeniowej, ziarnka (piksele), które opadną, zatrzymają się a co za tym idzie przestanie na nie oddziaływać grawitacja [Rys 1.].



[Rys. 1] Model opadania piasku, gdzie piksel dynamiczny oznacza ziarnko piasku, które nie zdążyło jeszcze opaść i zatrzymać się. Po zatrzymaniu piksel zmienia się w statyczny i przestaje na niego oddziaływać grawitacja.

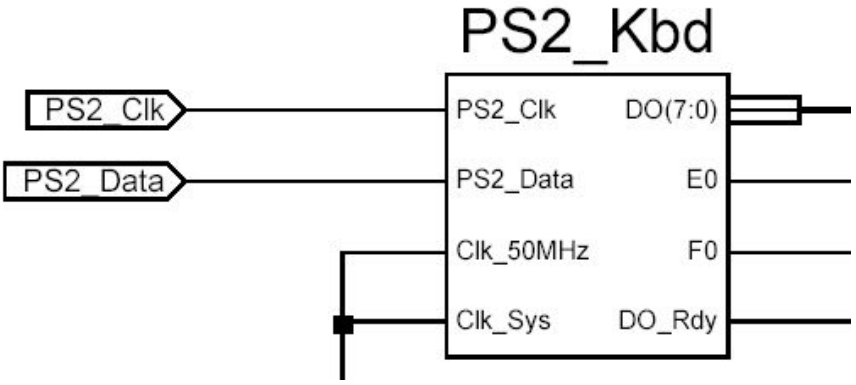
II. Realizacja projektu

W ramach projektu wykorzystaliśmy moduły dostępne na stronie internetowej projektu oraz stworzyliśmy własne moduły, które pozwoliły zrealizować etapy projektu. Podczas realizacji projektu natknęliśmy się spore problemy z optymalizacją naszej symulacji. Zakładaliśmy, że każdy z pikseli na ekranie będzie odpowiadał ziarenku piasku, niestety nie udało się zsyntezować tak dużej tablicy (około 300-400 tysięcy elementów). Przez to byliśmy zmuszeni do zwiększenia wielkości ziarenek piasku, każdy miał 8x8 pikseli, niestety to także dało ilość pikseli, której nie dało się zsyntezować przez brak miejsca na płytce (około 4 tysięcy elementów). Za trzecim razem ograniczyliśmy pole, w którym działa grawitacja oraz zwiększyliśmy ilość pikseli na ziarno do 16x16, co dało nam tablicę o 500 elementach. Finalnie udało nam się zsyntezować tablicę o wielkości 300 elementów, zajęło to około 7 minut.

III. Opis wykorzystywanych modułów

1. Moduł PS2_Kbd

Do obsługi klawiatury wykorzystaliśmy gotowy już moduł, który znacznie ułatwił i przyspieszył pracę oraz umożliwił nam skupienie się ważniejszych elementach takich jak wyświetlanie oraz symulacja samego piasku. Moduł był nam potrzebny ze względu na opisane w założeniach poruszanie się po danym obszarze i wybór miejsca, w którym piasek miał się pojawiać.



[Rys. 2] Schemat modułu PS2

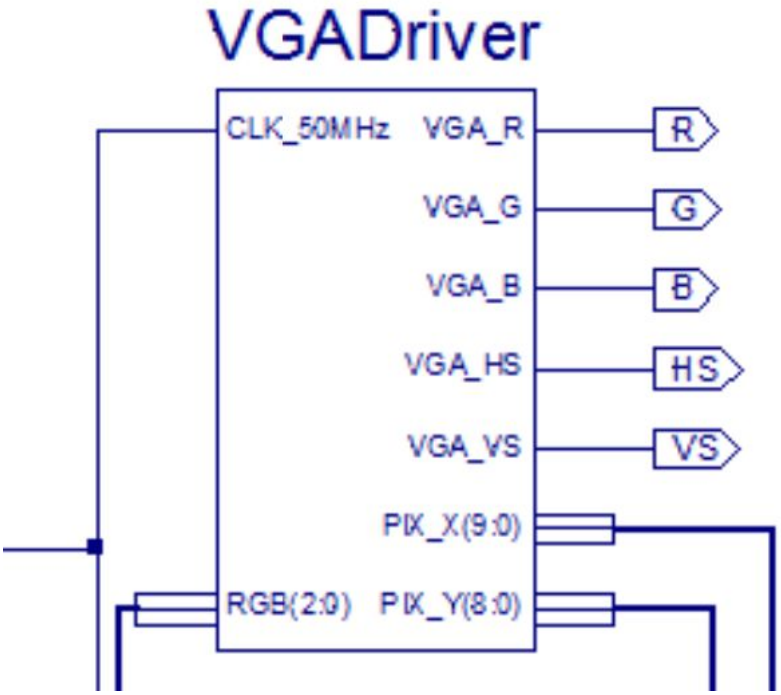
Wejścia:
PS2_Clk - zegar z zewnętrznego wyprowadzenia
PS2_Data - dane z zewnętrznego wyprowadzenia
Clk_50MHz - zegar
Clk_Sys - zegar

Wyjścia:
DO - kod przycisku
E0 - informacja czy kod był poprzedzony E0
F0 - informacja czy kod był poprzedzony F0
DO_Rdy - impulsy jednotaktowe sygnalizujące zakończenie odbioru kodu

2. Moduł VGADriver

(kod modułu na dole sprawozdania)

Do obsługi monitora VGA stworzyliśmy swój własny moduł. Wzorowaliśmy się na gotowym i działającym kodzie tworząc własny moduł spełniający nasze wymagania. Odpowiada on za wyświetlanie pikseli na ekranie w ośmu różnych kolorach. Wykorzytstaliśmy go do wyświetlania rezultatu obliczeń dokonywanych przez moduł symulacji piasku (PowderSimulator).



[Rys. 3] Schemat modułu VGA

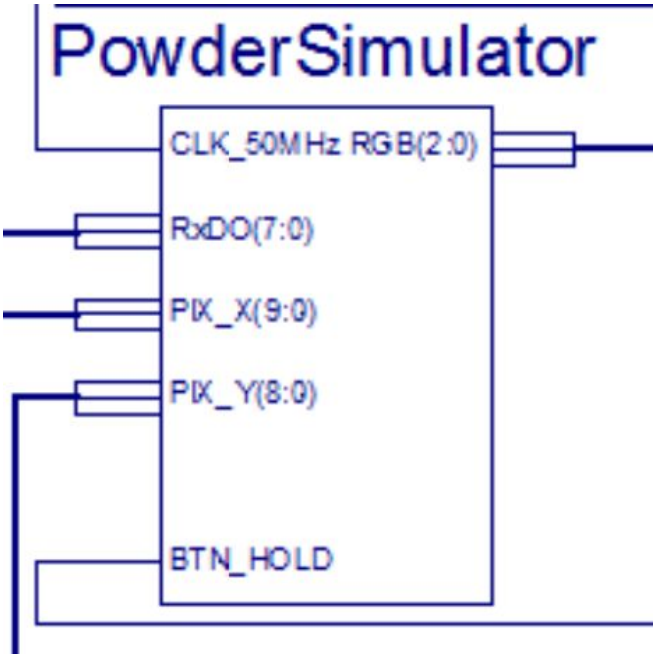
Wejścia:
CLK_50MHz - zegar
RGB - kolor danego piksela

Wyjścia:
VGA_R - kolor czerwony
VGA_G - kolor zielony
VGA_B - kolor niebieski
VGA_HS - synchronizacja pozioma
VGA_VS - synchronizacja pionowa
PIX_X - współrzędna x następnego piksela do “zabarwienia”
PIX_Y - współrzędna y następnego piksela do “zabarwienia”

3. Moduł PowderSimulator

(kod modułu na dole sprawozdania)

Moduł PowderSimulator stanowi rdzeń naszego projektu. Odbiera on i interpretuje w odpowiedni sposób sygnały dostarczone przez moduł obsługi klawiatury (PS2_Kbd), a następnie zależnie od wciśniętego przycisku, oblicza nową pozycję kursora (przemieszcza kursor) lub tworzy/usuwa w jego pozycji (kursora) ziarnka piasku. Ponadto przechowuje jednowymiarową tablicę typu boolean, która opisuje aktualny stan i położenie ziaren piasku. Najważniejszym jednak zadaniem tego modułu jest obliczanie fizyki ziaren piasku tak by oddziaływała na nie grawitacja, oraz by rozsypywały się na boki i tworzyły piaskowe “wzniesienia”. Silnik fizyki działa niezależnie od stanu klawiatury dlatego usunięcie/dodanie ziaren piasku z/na dowolnej pozycji nie spowoduje żadnych problemów. Aby rezultat obliczeń tego modułu był widoczny na ekranie przesyła on odpowiednie dane (w tym kolor) do modułu odpowiedzialnego za wyświetlanie pikseli na ekranie (VGADriver).

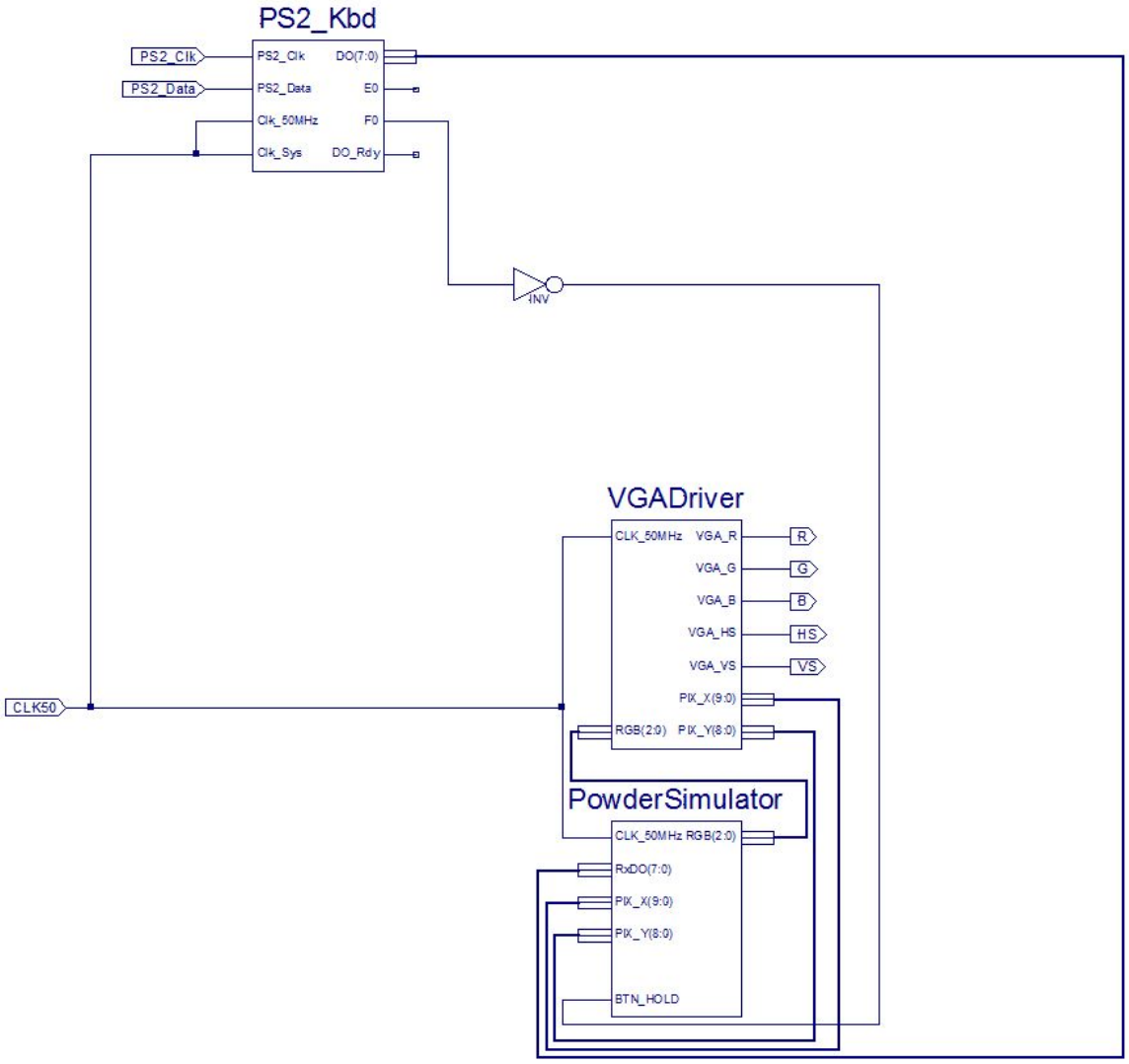


[Rys. 4] Schemat modułu Symulacji

Wejścia:
CLK_50MHz - zegar
RxDO - kod przycisku z modułu PS2
PIX_X - współrzędna x piksela z modułu VGA
PIX_Y - współrzędna y piksela z modułu VGA
BTN_HOLD - informacja o wciśniętym klawiszu z modułu PS2

Wyjścia:
RGB - informacja o kolorze dla modułu VGA

IV. Kompletny schemat połączeń między modułami

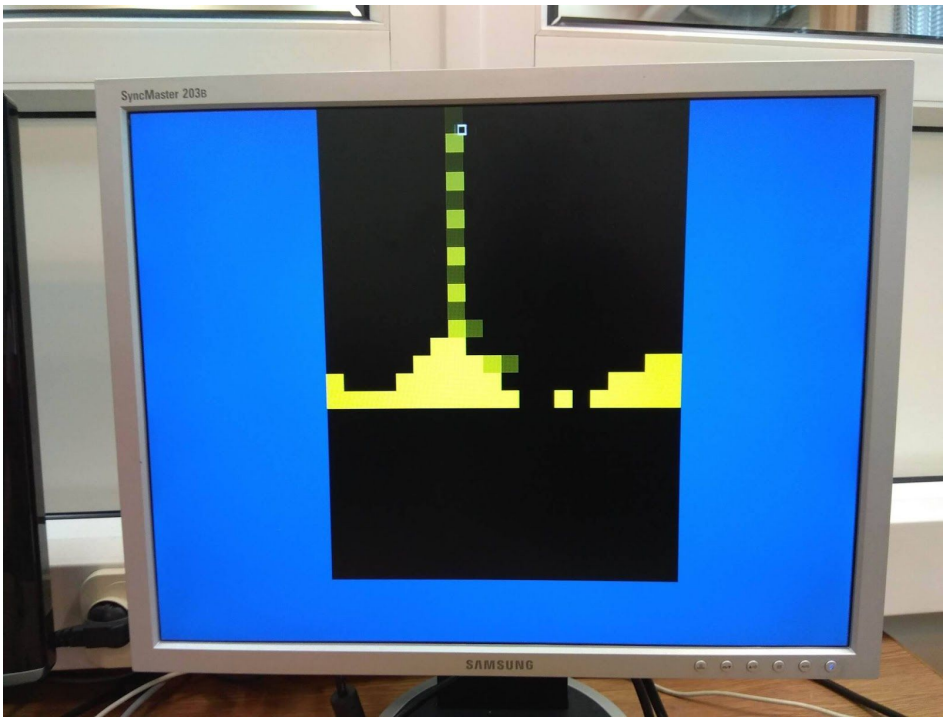


[Rys. 5] Schemat projektu przedstawiający wszystkie wykorzystane moduły i połączenia sygnałów między nimi.

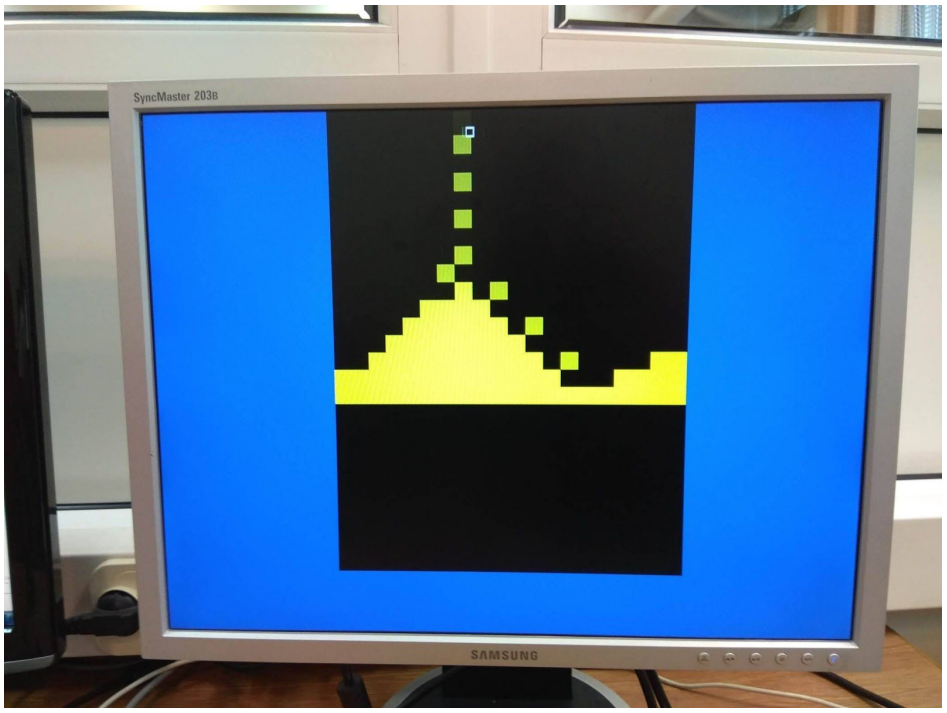
V. Instrukcja obsługi aplikacji

Po włączeniu programu na ekranie pokazuje się czarna przestrzeń ograniczona niebieskimi polami z każdej strony, jest to obszar, w którym działa symulacja fizyki piasku. Na ekranie przy pomocy klawiatury PS2 podłączonej do płytki można wykonać trzy operacje:

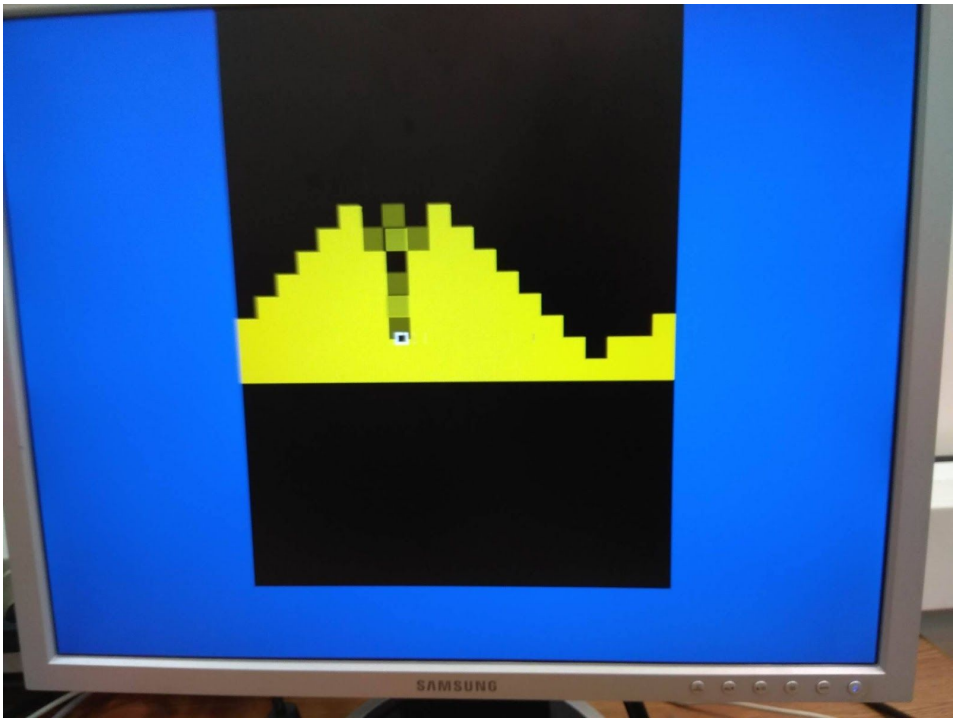
- Poruszać się kursorem:
    - Przycisk [W] - kursor porusza się w górę.
    - Przycisk [A] - kursor porusza się w lewo.
    - Przycisk [S] - kursor porusza się w dół.
    - Przycisk [D] - kursor porusza się w prawo.
- Tworzyć piasek przyciskiem [F].
- Usuwać piasek przyciskiem [X].



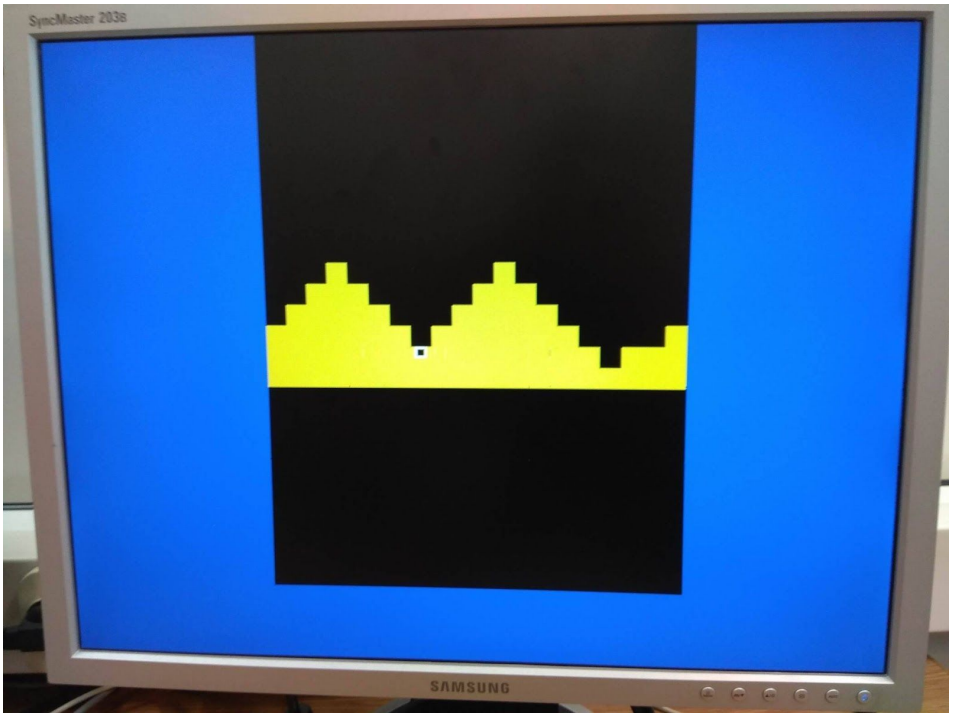
[Rys. 6] Tworzenie piasku poprzez przytrzymanie przycisku [F].



[Rys. 7] Tworzenie piasku poprzez przytrzymanie przycisku [F], parę sekund po Rys. 6.



[Rys. 8] Usuwanie piasku poprzez przytrzymanie przycisku [X].



[Rys. 9] Usuwanie piasku poprzez przytrzymanie przycisku [X], parę sekund po Rys. 8.

VI. Podsumowanie

Zaczynając pracę nad projektem nie myśleliśmy, że ograniczenia spowodowane językiem VHDL oraz samą płytką tak bardzo wpłyną na wynik końcowy i czas realizacji projektu. Wielokrotnie przez restrykcje języka VHDL oraz bardzo małą ilość pamięci wirtualnej w płytce, musieliśmy wymyślać dodatkowe mechanizmy optymalizacyjne, aby stopniowo zbliżać się do ustalonego na początku celu. Dodatkowo szybkość komputerów w pracowni oraz zasoby płytki nie pozwoliły nam na stworzenie symulacji dużej ilości ziaren piasku. Wraz z kolejnymi godzinami pracy nad projektem wielkość ziaren była powiększana, a ich ilość na ekranie znacznie zmniejszana. W efekcie końcowym początkowa tablica przechowująca stan ziaren piasku została zmniejszona z ponad 4000 elementów do niecałych 600 w tym tylko 300 ziaren, na które oddziaływała fizyka. Mimo tych problemów projekt zakończył się sukcesem, a główny cel projektu został osiągnięty. Symulacja - niestety w mniejszej skali - działa zaskakująco dobrze, a wynik końcowy projektu (mimo wszystko) jest niesamowicie satysfakcjonujący.

VII. Kod (własnych) modułów użytych w projekcie:

1. Moduł VGADriver

```
-- Załączanie odpowiednich bibliotek
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Definicja klasy modułu
entity VGADriver is
    Port ( CLK_50MHz : in  STD_LOGIC;
           RGB : in  STD_LOGIC_VECTOR (2 downto 0);
           VGA_R, VGA_G, VGA_B: out STD_LOGIC;
           VGA_HS, VGA_VS : out  STD_LOGIC;
           PIX_X : out  STD_LOGIC_VECTOR (9 downto 0);
           PIX_Y : out  STD_LOGIC_VECTOR (8 downto 0));
end VGADriver;

-- Definicja architektury modułu
architecture Behavioral of VGADriver is
    signal clk_25 : STD_LOGIC := '0';
    signal h_cnt : integer := 0;
    signal v_cnt : integer := 32;
    signal color : STD_LOGIC_VECTOR(11 downto 0);
begin

-- Dwukrotne (sztuczne) zmniejszanie częstotliwości zegara CLK_50MHz
    clk_div : process(CLK_50MHz)
    begin
        if (rising_edge(CLK_50MHz)) then clk_25 <= not clk_25; end if;
    end process clk_div;

-- Obliczanie indeksów wszystkich pikseli na ekranie
    counters : process(clk_25)
    begin
        if (rising_edge(clk_25)) then
            if (h_cnt < 799) then h_cnt <= h_cnt + 1;
            else h_cnt <= 0;
                if (v_cnt < 520) then v_cnt <= v_cnt + 1;
                else v_cnt <= 0; end if;
            end if;
        end if;
    end process counters;
```

```
-- Synchronizacja wartości obliczonych wcześniej indeksów (liczby w instrukcjach
warunkowych pochodzą z gotowych tablic).
    h_sync : process(h_cnt)
    begin
      if (h_cnt < 96) then VGA_HS <= '0';
      else VGA_HS <= '1'; end if;
    end process h_sync;

    v_sync : process(v_cnt)
    begin
      if (v_cnt < 2) then VGA_VS <= '0';
      else VGA_VS <= '1'; end if;
    end process v_sync;

   -- Kolorowanie pikseli na odpowiednich pozycjach na odpowiedni kolor
(o ile piksele znajdują się w zasięgu ekranu).
    color_pixel : process(h_cnt, v_cnt, RGB)
    begin
        if (h_cnt >= 144 and h_cnt <= 783 and
        v_cnt >= 31 and v_cnt <= 510) then

            VGA_R <= RGB(2); VGA_G <= RGB(1); VGA_B <= RGB(0);

            PIX_X <= std_logic_vector(to_unsigned(h_cnt - 144, 10));
            PIX_Y <= std_logic_vector(to_unsigned(v_cnt - 31, 9));
        else
            VGA_R <= '0'; VGA_G <= '0'; VGA_B <= '0';

            PIX_X <= std_logic_vector(to_unsigned(640, 10));
            PIX_Y <= std_logic_vector(to_unsigned(480, 9));
        end if;
    end process color_pixel;

end Behavioral;
```

2. Moduł PowderSimulator

```
-- Załączanie potrzebnych bibliotek
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Definicja klasy modułu
entity PowderSimulator is
    Port ( CLK_50MHz : in STD_LOGIC;
           RxDO : in  STD_LOGIC_VECTOR (7 downto 0);
           PIX_X : in  STD_LOGIC_VECTOR (9 downto 0);
           PIX_Y : in  STD_LOGIC_VECTOR (8 downto 0);
           BTN_HOLD : in STD_LOGIC;
           RGB : out  STD_LOGIC_VECTOR (2 downto 0));
end PowderSimulator;

-- Definicja architektury modułu
architecture Behavioral of PowderSimulator is

-- Definiowanie sygnałów (związanych z kursorem)
signal cursor_x : integer := 100;
signal cursor_y : integer := 100;
signal clock_counter : integer := 0;
signal clk_main : STD_LOGIC := '0';
signal clock_grav_counter : integer := 0;
signal clk_grav : STD_LOGIC := '0';
signal conv_pix_x : integer := 0;
signal conv_pix_y : integer := 0;
signal x_i : integer := 0;
signal y_i : integer := 0;
signal f_i : integer := 0;
signal xc_i : integer := 0;
signal yc_i : integer := 0;
signal fc_i : integer := 0;

-- Definiowanie sygnałów (związanych z silnikiem fizyki ziaren piasku)
signal put_sand_flag : STD_LOGIC := '0';
signal remove_sand_flag : boolean := false;
signal put_sand_flag_proc : STD_LOGIC := '0';
signal put_sand_idx : integer := 0;
signal y_plus_idx : integer := 0;
signal grav_i : integer := 0;
signal temp_check : STD_LOGIC := '0';
signal sandSide : boolean := false;
signal isLeftEmpty : boolean := false;
signal isRightEmpty : boolean := false;
signal isSandMoved : boolean := false;

-- Definiowanie sygnałów (związanych ze strumieniem wejścia z klawiatury)
signal key_w : STD_LOGIC := '0';
signal key_s : STD_LOGIC := '0';
signal key_a : STD_LOGIC := '0';
signal key_d : STD_LOGIC := '0';
signal key_f : STD_LOGIC := '0';
signal key_x : STD_LOGIC := '0';
```

```
-- Definiowanie tablicy przechowującej stan ziaren piasku
type pixel_array is array (integer range <> ) of boolean;
signal pixel_screen : pixel_array (519 downto 0);

begin
   -- Tworzenie dwóch (sztucznych) zegarów o różnych częstotliwościach
   clk_div : process(CLK_50MHz)
   begin
        if (rising_edge(CLK_50MHz)) then
            -- Zegar odpowiedzialny za częstotliwość odczytywania stanu klawiatury
            clock_counter <= clock_counter + 1;
            if( clock_counter > 200000 ) then
               clock_counter <= 0;
               clk_main <= not clk_main;
            end if;
            -- Zegar odpowiedzialny za akutalizowanie silinika fizyki ziaren
            clock_grav_counter <= clock_grav_counter + 1;
            if( clock_grav_counter > 1000000 ) then
               clock_grav_counter <= 0;
               clk_grav <= not clk_grav;
            end if;
        end if;
   end process clk_div;

   -- Analizowanie wciśniętych klawiszy
   input : process(BTN_HOLD, RxDO)
   begin

      if(BTN_HOLD = '0') then
         if(RxDO = "00011101") then key_w <= '0';
         elsif(RxDO = "00011011") then key_s <= '0';
         elsif(RxDO = "00011100") then key_a <= '0';
         elsif(RxDO = "00100011") then key_d <= '0';
         elsif(RxDO = "00101011") then key_f <= '0';
         elsif(RxDO = "00100010") then key_x <= '0';
         else
            key_w <= '0';
            key_s <= '0';
            key_a <= '0';
            key_d <= '0';
            key_f <= '0';
            key_x <= '0';
         end if;
      elsif(BTN_HOLD = '1') then
         if(RxDO = "00011101") then key_w <= '1';
         elsif(RxDO = "00011011") then key_s <= '1';
         elsif(RxDO = "00011100") then key_a <= '1';
         elsif(RxDO = "00100011") then key_d <= '1';
         elsif(RxDO = "00101011") then key_f <= '1';
         elsif(RxDO = "00100010") then key_x <= '1'; end if;
      end if;
   end process input;
```

```
-- Wywoływanie odpowiednich funkcji na podstawie wciśniętych klawiszy
cursor : process(clk_main)
   begin
      if(rising_edge(clk_main)) then
         put_sand_flag <= '0';
         remove_sand_flag <= false;
         if(key_w = '1') then -- Ruch kursora do góry
            cursor_y <= cursor_y - 1;
            if (cursor_y < 5) then cursor_y <= 5; end if;
         elsif(key_s = '1') then -- Ruch kursora w dół
            cursor_y <= cursor_y + 1;
            if(cursor_y > 470) then cursor_y <= 470; end if;
         end if;
         if(key_a = '1') then -- Ruch kursora w lewo
            cursor_x <= cursor_x - 1;
            if(cursor_x < 160) then cursor_x <= 160; end if;
         elsif(key_d = '1') then -- Ruch kursora w prawo
            cursor_x <= cursor_x + 1;
            if(cursor_x > 470) then cursor_x <= 470; end if;
         elsif(key_f = '1') then -- Tworzenie ziaren piasku
            xc_i <= (cursor_x - 5) / 16 - 10;
            yc_i <= (cursor_y - 5) / 16;
            fc_i <= xc_i + 20 * yc_i;
            put_sand_idx <= fc_i; -- Indeks pozycji, na której
                                  ma zostać stworzone ziarno
            put_sand_flag <= '1';
         elsif(key_x = '1') then -- Usuwanie ziaren piasku
            xc_i <= (cursor_x - 5) / 16 - 10;
            yc_i <= (cursor_y - 5) / 16;
            fc_i <= xc_i + 20 * yc_i;
            put_sand_idx <= fc_i; -- Indeks pozycji, na której
                                  ma zostać usunięte ziarno
            remove_sand_flag <= true;
         end if;
      end if;
end process cursor;
```

```
-- Symulowanie zachowania piasku (grawitacja i opadanie ziaren na boki)
gravity : process(clk_grav)
   begin
      if(rising_edge(clk_grav)) then
         if(put_sand_flag = '1') then -- tworzenie nowych ziaren
            pixel_screen(put_sand_idx) <= true;
         elsif(remove_sand_flag = true) then -- usuwanie ziaren
            pixel_screen(put_sand_idx) <= false;
         end if;
         -- Symulowanie fizyki dla 300 ziaren
            (zmiana tej wartości znacząco wpływa na czas syntezy)
         for i in 299 downto 0 loop
            -- Nie ma sensu obliczać fizyki dla pustych pól (bez ziaren)
            if (pixel_screen(i) = true) then
               -- Sprawdzanie czy pod ziarnem jest puste pole
               if (pixel_screen(i + 20) = false) then
                  pixel_screen(i) <= false;
                  pixel_screen(i + 20) <= true;
               else
                  -- Sprawdzanie czy ziarno może przesunąć się w lewo
                  if (i > 1 and pixel_screen(i - 1) = false and
                   pixel_screen(i + 19) = false) then
                     pixel_screen(i) <= false;
                     pixel_screen(i - 1) <= true;
                  -- Sprawdzanie czy ziarno może przesunąć się w prawo
                  elsif (pixel_screen(i + 1) = false and
                   pixel_screen(i + 21) = false) then
                     pixel_screen(i) <= false;
                     pixel_screen(i + 1) <= true;
                  end if;
               end if;
            end if;
         end loop;
      end if;
end process gravity;
```

```
   -- Wyświetlanie rezultatu obliczeń na ekranie
   render : process(PIX_X, PIX_Y)
      begin
         -- Obliczanie indeksu w jednowymiarowej tablicy odpowiadającego aktualnej
pozycji kursora
         conv_pix_x <= to_integer(unsigned(PIX_X));
         conv_pix_y <= to_integer(unsigned(PIX_Y));
         x_i <= (conv_pix_x - 5) / 16 - 10;
         y_i <= (conv_pix_y - 5) / 16;
         f_i <= x_i + 20 * y_i;

         -- Renderowanie czarnego wnętrza dla kursora
         if (cursor_x + 2 < conv_pix_x and cursor_x + 8 > conv_pix_x and cursor_y
+ 2 < conv_pix_y and cursor_y + 8 > conv_pix_y) then
            RGB <= "000";
         -- Renderowanie białej otoczki dla kursora
         elsif (cursor_x < conv_pix_x and cursor_x + 10 > conv_pix_x and cursor_y
< conv_pix_y and cursor_y + 10 > conv_pix_y) then
            RGB <= "111";
         -- Renderowanie piasku
         elsif (x_i >= 0 and x_i < 20 and y_i >= 0 and y_i < 26) then
            if (pixel_screen(f_i) = true) then
               RGB <= "110"; -- Piasek
            else
               RGB <= "000"; -- Puste pole (bez piasku)
            end if;
         else
            RGB <= "001"; -- Renderowanie niebieskiej przestrzeni
         end if;
   end process render;

end Behavioral;
```

VIII. Wykorzystane źródła

- http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/uc/
- http://tinyvga.com/vga-timing
- https://www.digikey.com/eewiki/pages/viewpage.action?pageId=15925278